Lilliput

Compressed Object Headers

Roman Kennke (@rkennke)

Principal Engineer
Amazon

Thomas Stüfe (@tstuefe)

Principal Engineer
Red Hat

Agenda

Overview/Motivation

Introduction

Locking

GC Forwarding

Compressed Class Pointers

Overview/Motivation

What is Project Lilliput?

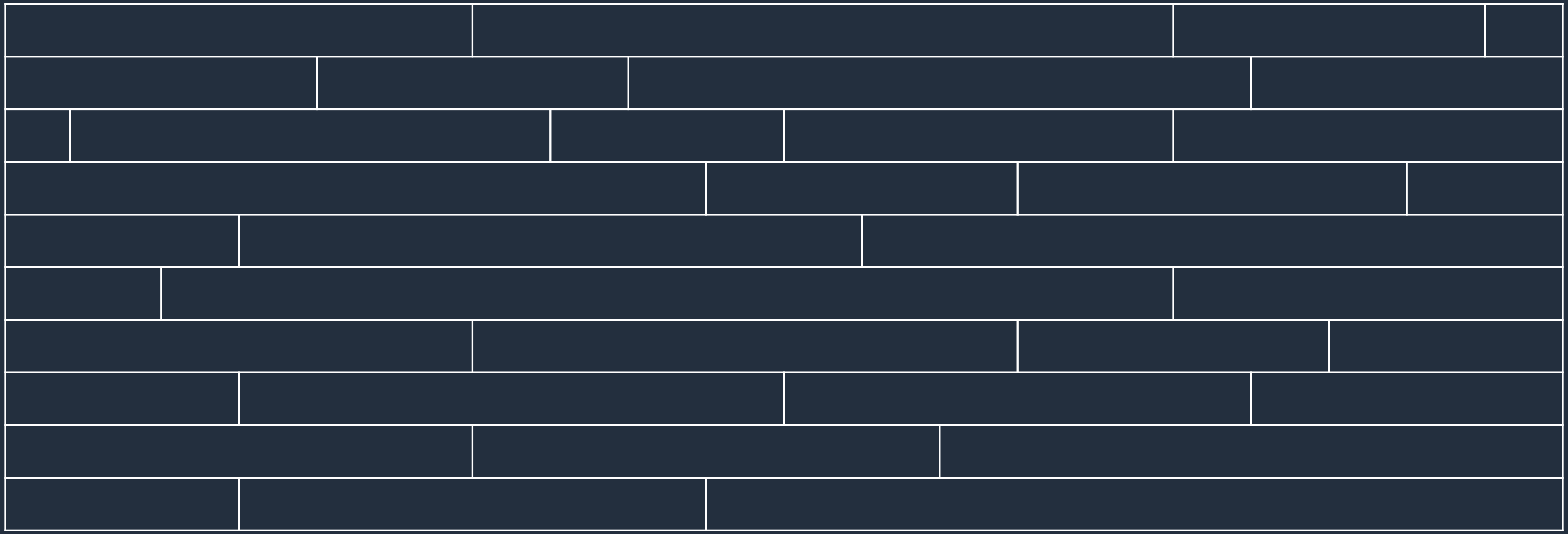
- An OpenJDK project
  (Contributions by: Red Hat, Oracle, SAP, Huawei, Alibaba, Amazon, …)

- Goal: Reduce memory footprint

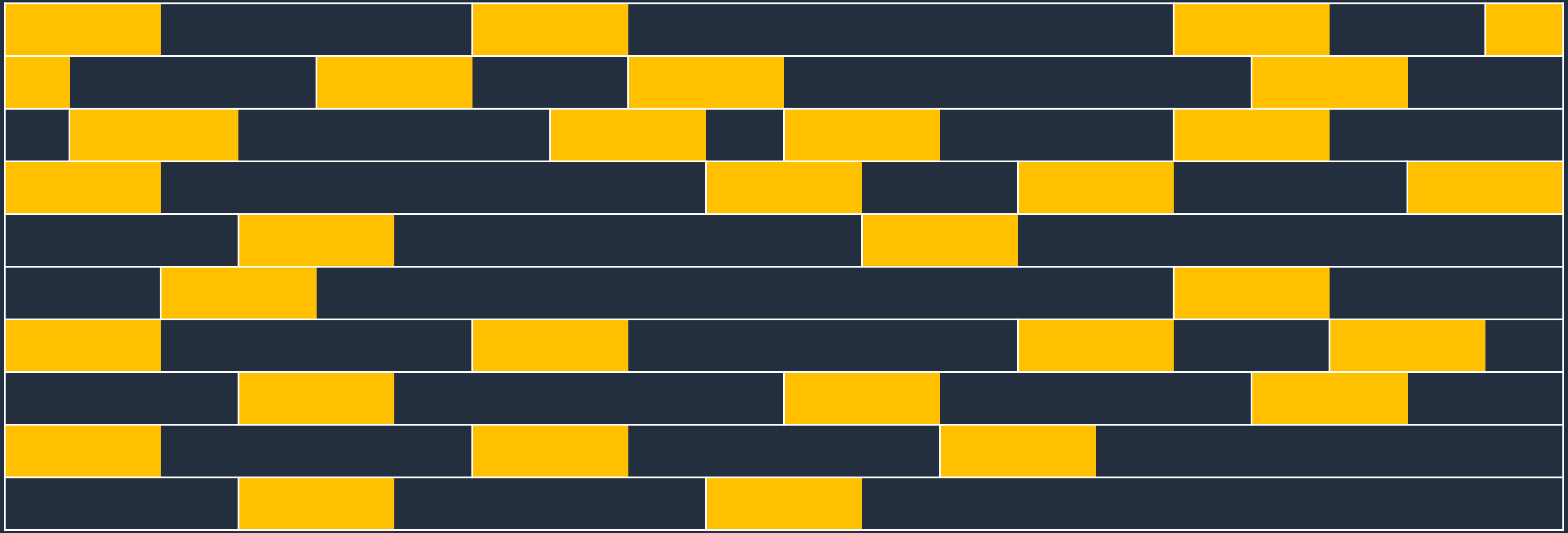
  Side-effects: potential CPU and latency improvements

- Specifically: Reduce size of (Java) object headers

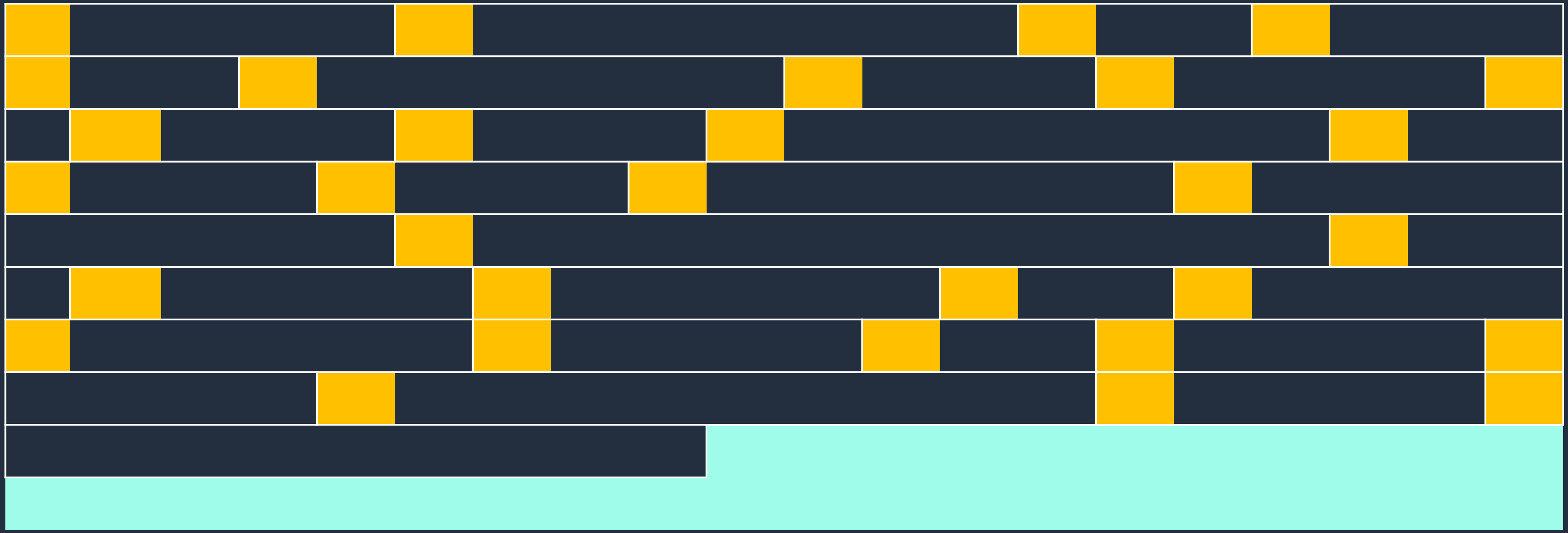
Motivation



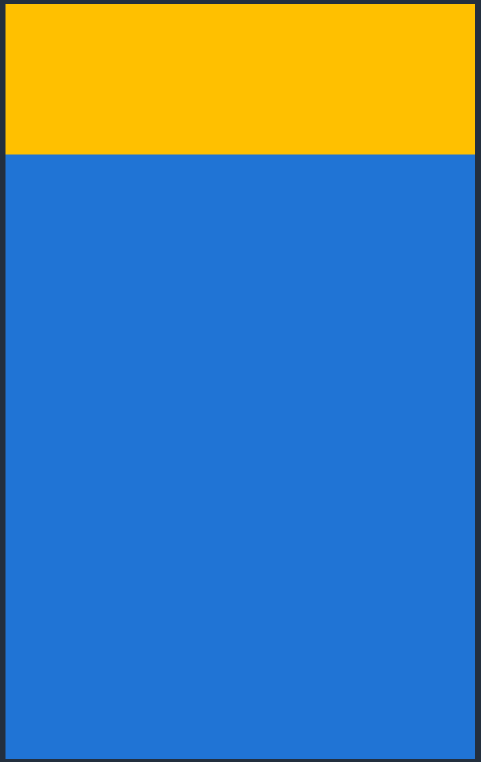
Motivation



Motivation

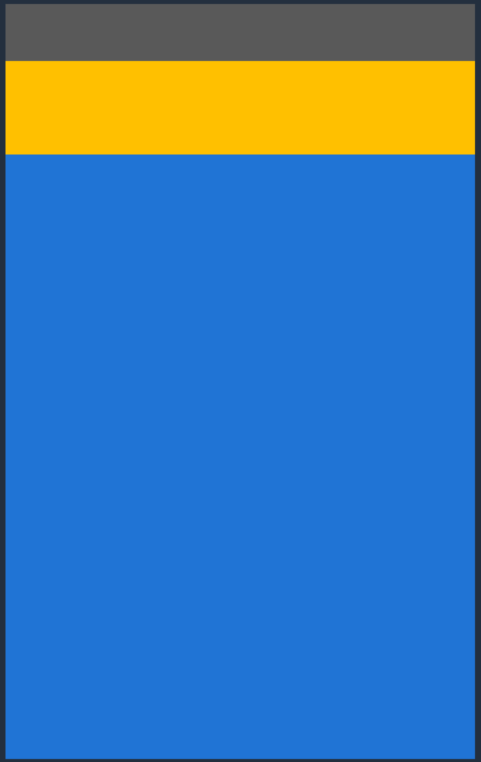


Motivation – 12-bytes headers



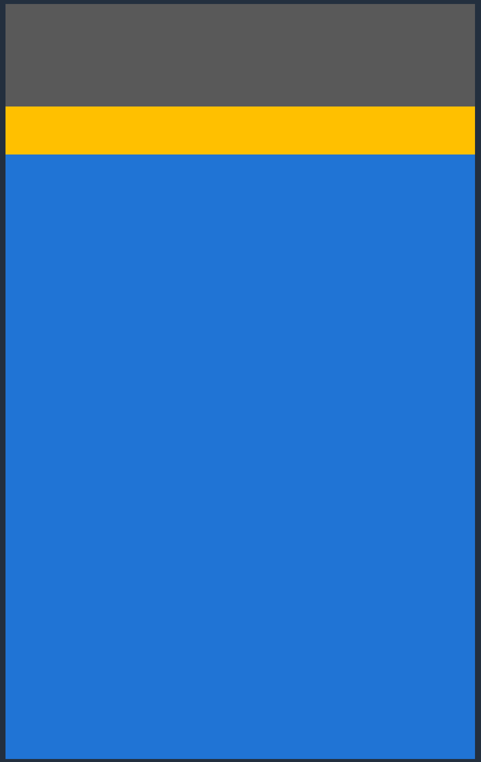
- ~20% of live data on heap is object header
- YMMV (0% - 50%)

Motivation – 8-bytes headers



- ~13% of live data on heap is object header
- YMMV (0% - 33%)
- Average savings of 7% (up to ~30%)

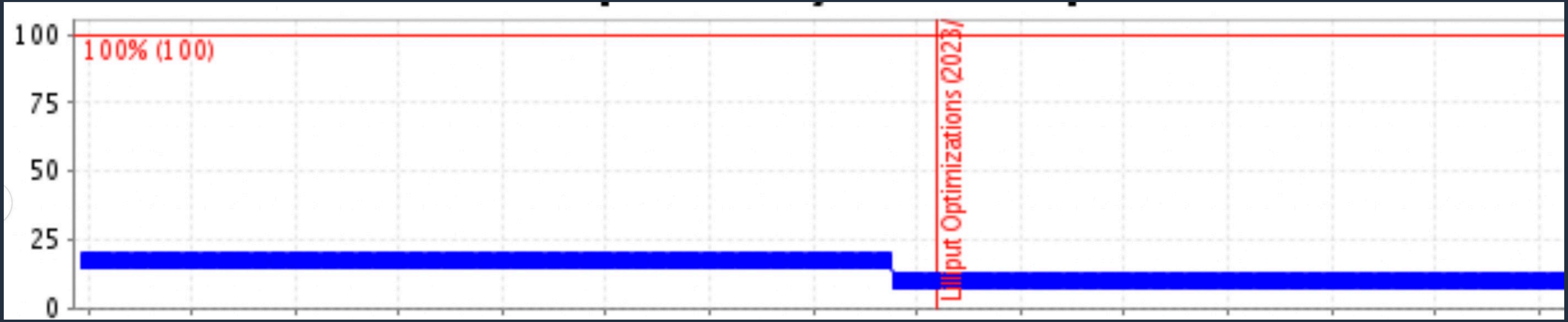
Motivation – 4-bytes headers



- ~6.7% of live data on heap is object header
- YMMV (0% - 17%)
- Average savings of 14% (up to ~50%)

Motivation

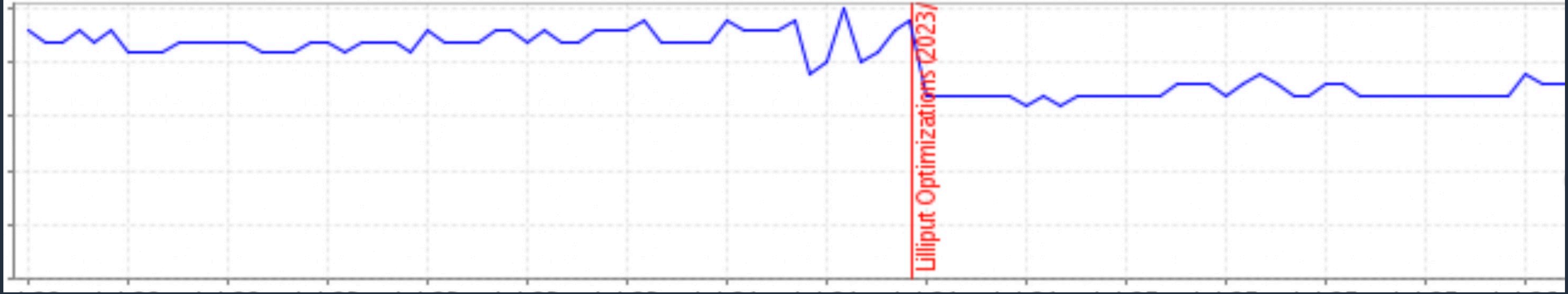
Heap usage after GC



Down by >30%

Motivation

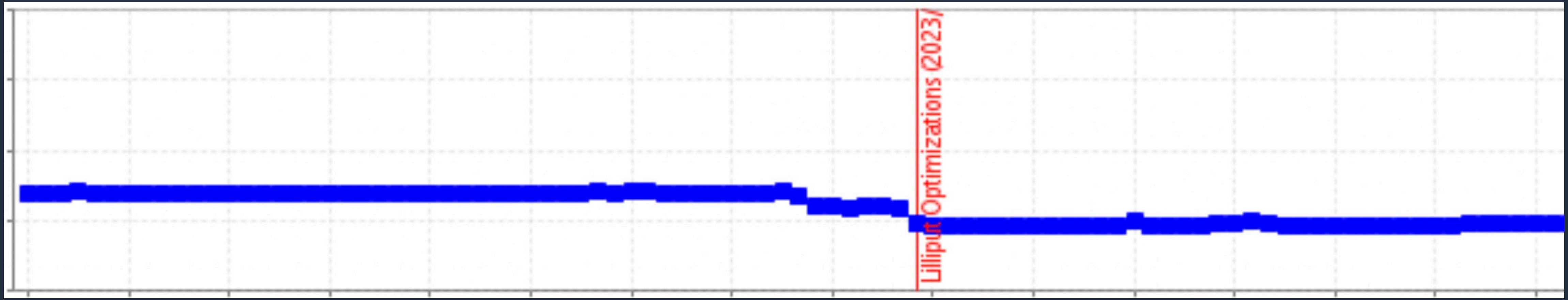
CPU Utilization



Down by ~25%

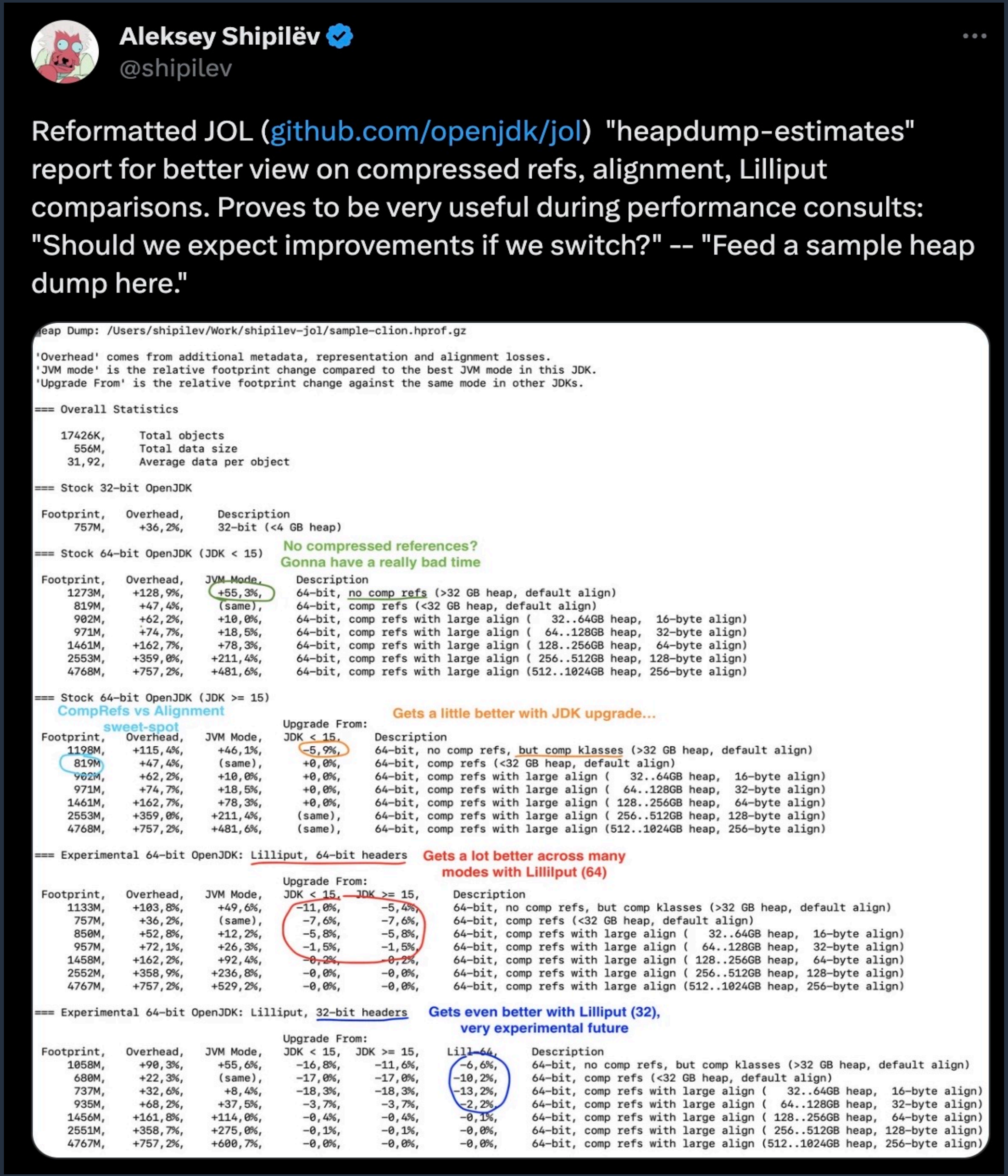
Motivation

Latency



Down by ~30%

JOL



Aleksey Shipilëv @shipilev

Reformatted JOL (github.com/openjdk/jol) "heapdump-estimates" report for better view on compressed refs, alignment, Lilliput comparisons. Proves to be very useful during performance consults: "Should we expect improvements if we switch?" -- "Feed a sample heap dump here."

```
Heap Dump: /Users/shipilev/Work/shipilev-jol/sample-clion.hprof.gz

'Overhead' comes from additional metadata, representation and alignment losses.
'JVM mode' is the relative footprint change compared to the best JVM mode in this JDK.
'Upgrade From' is the relative footprint change against the same mode in other JDKs.

=== Overall Statistics

   17426K,     Total objects
     556M,     Total data size
    31,92,     Average data per object

=== Stock 32-bit OpenJDK

 Footprint,   Overhead,     Description
      757M,     +36,2%,     32-bit (<4 GB heap)

=== Stock 64-bit OpenJDK (JDK < 15)
```

No compressed references?
Gonna have a really bad time

```
 Footprint,   Overhead,   JVM Mode,     Description
     1273M,    +128,9%,    +55,3%,      64-bit, no comp refs (>32 GB heap, default align)
      819M,     +47,4%,    (same),      64-bit, comp refs (<32 GB heap, default align)
      902M,     +62,2%,    +10,0%,      64-bit, comp refs with large align (   32..64GB heap,  16-byte align)
      971M,     +74,7%,    +18,5%,      64-bit, comp refs with large align (  64..128GB heap,  32-byte align)
     1461M,    +162,7%,    +78,3%,      64-bit, comp refs with large align ( 128..256GB heap,  64-byte align)
     2553M,    +359,0%,   +211,4%,      64-bit, comp refs with large align ( 256..512GB heap, 128-byte align)
     4768M,    +757,2%,   +481,6%,      64-bit, comp refs with large align (512..1024GB heap, 256-byte align)

=== Stock 64-bit OpenJDK (JDK >= 15)
```

CompRefs vs Alignment
sweet-spot

Gets a little better with JDK upgrade...

```
                                      Upgrade From:
 Footprint,   Overhead,   JVM Mode,   JDK < 15,     Description
     1198M,    +115,4%,    +46,1%,      -5,9%,      64-bit, no comp refs, but comp klasses (>32 GB heap, default align)
      819M,     +47,4%,    (same),      +0,0%,      64-bit, comp refs (<32 GB heap, default align)
      902M,     +62,2%,    +10,0%,      +0,0%,      64-bit, comp refs with large align (   32..64GB heap,  16-byte align)
      971M,     +74,7%,    +18,5%,      +0,0%,      64-bit, comp refs with large align (  64..128GB heap,  32-byte align)
     1461M,    +162,7%,    +78,3%,      +0,0%,      64-bit, comp refs with large align ( 128..256GB heap,  64-byte align)
     2553M,    +359,0%,   +211,4%,     (same),      64-bit, comp refs with large align ( 256..512GB heap, 128-byte align)
     4768M,    +757,2%,   +481,6%,     (same),      64-bit, comp refs with large align (512..1024GB heap, 256-byte align)

=== Experimental 64-bit OpenJDK: Lilliput, 64-bit headers
```

Gets a lot better across many
modes with Lillilput (64)

```
                                      Upgrade From:
 Footprint,   Overhead,   JVM Mode,   JDK < 15,   JDK >= 15,     Description
     1133M,    +103,8%,    +49,6%,     -11,0%,      -5,4%,       64-bit, no comp refs, but comp klasses (>32 GB heap, default align)
      757M,     +36,2%,    (same),      -7,6%,      -7,6%,       64-bit, comp refs (<32 GB heap, default align)
      850M,     +52,8%,    +12,2%,      -5,8%,      -5,8%,       64-bit, comp refs with large align (   32..64GB heap,  16-byte align)
      957M,     +72,1%,    +26,3%,      -1,5%,      -1,5%,       64-bit, comp refs with large align (  64..128GB heap,  32-byte align)
     1458M,    +162,2%,    +92,4%,      -0,2%,      -0,2%,       64-bit, comp refs with large align ( 128..256GB heap,  64-byte align)
     2552M,    +358,9%,   +236,8%,      -0,0%,      -0,0%,       64-bit, comp refs with large align ( 256..512GB heap, 128-byte align)
     4767M,    +757,2%,   +529,2%,      -0,0%,      -0,0%,       64-bit, comp refs with large align (512..1024GB heap, 256-byte align)

=== Experimental 64-bit OpenJDK: Lilliput, 32-bit headers
```

Gets even better with Lilliput (32),
very experimental future

```
                                      Upgrade From:
 Footprint,   Overhead,   JVM Mode,   JDK < 15,   JDK >= 15,    Lill-64,     Description
     1058M,     +90,3%,    +55,6%,     -16,8%,     -11,6%,      -6,6%,      64-bit, no comp refs, but comp klasses (>32 GB heap, default align)
      680M,     +22,3%,    (same),     -17,0%,     -17,0%,     -10,2%,      64-bit, comp refs (<32 GB heap, default align)
      737M,     +32,6%,     +8,4%,     -18,3%,     -18,3%,     -13,2%,      64-bit, comp refs with large align (   32..64GB heap,  16-byte align)
      935M,     +68,2%,    +37,5%,      -3,7%,      -3,7%,      -2,2%,      64-bit, comp refs with large align (  64..128GB heap,  32-byte align)
     1456M,    +161,8%,   +114,0%,      -0,4%,      -0,4%,      -0,1%,      64-bit, comp refs with large align ( 128..256GB heap,  64-byte align)
     2551M,    +358,7%,   +275,0%,      -0,1%,      -0,1%,      -0,0%,      64-bit, comp refs with large align ( 256..512GB heap, 128-byte align)
     4767M,    +757,2%,   +600,7%,      -0,0%,      -0,0%,      -0,0%,      64-bit, comp refs with large align (512..1024GB heap, 256-byte align)
```

Motivation

- Reduce hardware (or cloud) cost
- Drive more load
- Reduce energy bills
- Save CO2

Introduction

What’s in it?

```
Mark Word (normal):

 64                     39                              8.   3  0
  [.......................HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH.AAAA.TT]
           (Unused)                (Hash Code)        (GC Age)(Tag)
```

```
Class Word (compressed):
 32                               0
  [CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC]
      (Compressed Class Pointer)
```

Insight:

- Most objects never get i-hashed
- Most objects never get locked

What’s in it?

```
Mark Word (overloaded):

 64                                                           2 0
  [ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppTT]
                     (Native Pointer)                        (Tag)


Class Word (compressed):
 32                               0
  [CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC]
      (Compressed Class Pointer)
```

What’s in it?

```
Mark Word (overloaded):

 64                                                            2 0
  [ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppTT]
                      (Native Pointer)                         (Tag)
```

-> Pointer into stack (for stack-locking) (tag = 00)

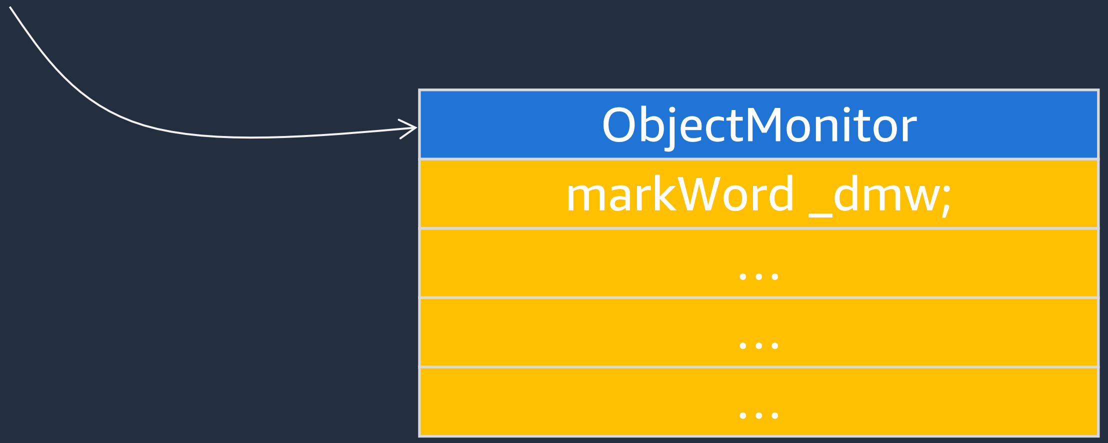
-> Pointer to ObjectMonitor (for monitor-locking) (tag = 10)

-> Pointer to forwarded object (for GC forwarding) (tag = 11)

Displaced mark-word

```
Mark Word (overwritten):

 64                                                            2 0
  [ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp10]
                      (Native Pointer)                        (Tag)
```



The Plan

```
Header (compact):
 64                              32                        7   3  0
  [CCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHuuuuAAAASTT]
     (Compressed Class Pointer)             (Hash Code) (GC Age)^(Tag)
                                                 (Self Forwarded Tag)
```

The Plan – Lilliput 2

```
Header (Lilliput 2):
 32                      9 7   3  0
  [CCCCCCCCCCCCCCCCCCCCCCCHHAAAASTT]
       (Class Pointer)    ^(Age)^(Tag)
                 (Hash-Code) (Self Forwarded Tag)
```

The Problems

- Old:
  - Header rarely carries ‘interesting’ information (locked, i-hashed)
  - Class-pointer is in separate field which never gets touched
- New:
  - Class-pointer is part of header
  - Must never loose that pointer
  - Header displacement and GC forwarding overwrite header

The Problems

- How to fit everything into fewer bits?
- How to safely access header when displaced?
- How to avoid clobbering the class-pointer?

Locking

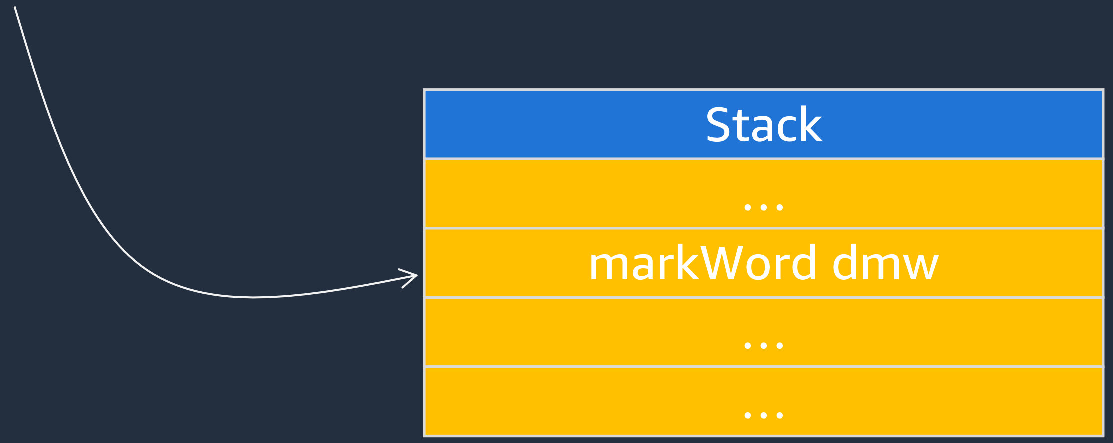
Stack-Locking

- Simplest locking primitive
- Coordinate threads by CAS-ing on object mark-word
- No contention
- No support for wait()/notify()
- No support for JNI
- -> Inflate to full ObjectMonitor

Stack-locking

```
Mark Word (stack-locked):

 64                                                           2 0
  [pppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp00]
                    (Stack-Pointer)                          (Tag)
```

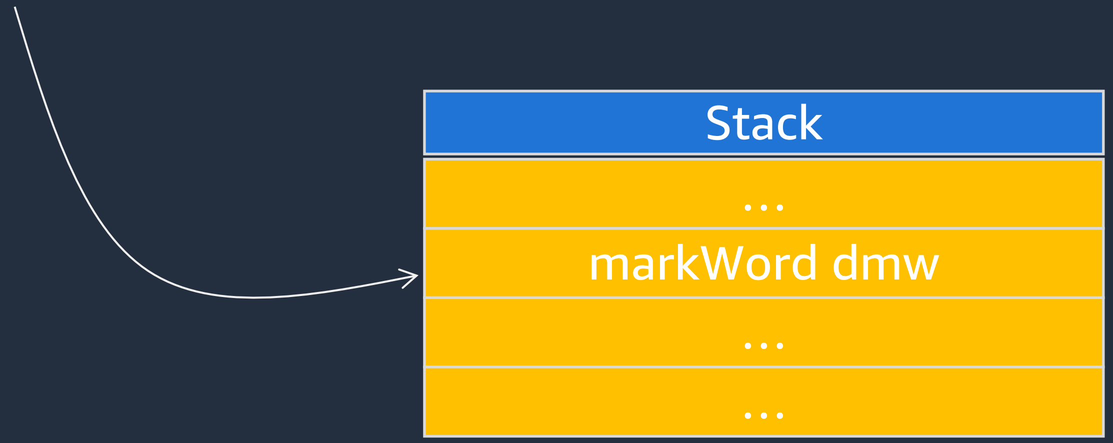


Is Thread T locking object O?
(Not: Which thread is locking O?)

Stack-locking

```
Mark Word (stack-locked):

 64                                                         2 0
  [pppppppppppppppppppppppppppppppppppppppppppppppppppppppppp00]
                    (Stack-Pointer)                       (Tag)
```



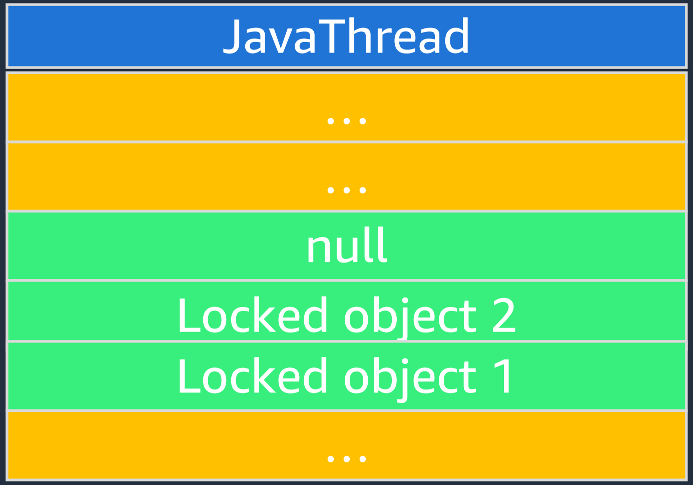
Accessing dmw is dangerous!

(racy with unlocking)

New lightweight locking

```
Header (lw-locked):
 64                              32                        7    3 0
  [CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHHHHHHHHHHAAAAS00]
     (Compressed Class Pointer)             (Hash Code) (GC Age)^(Tag)
                                                 (Self Forwarded Tag)
```

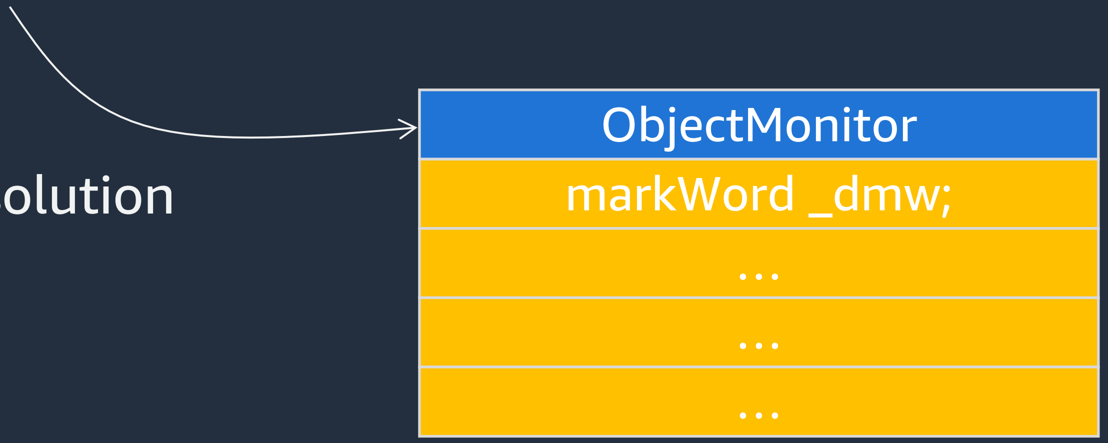
Is Thread T locking object O?
(Not: Which thread is locking O?)



Monitor locking

```
Mark Word (overwritten):

 64                                                            2 0
  [pppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp10]
                     (Native Pointer)                         (Tag)
```



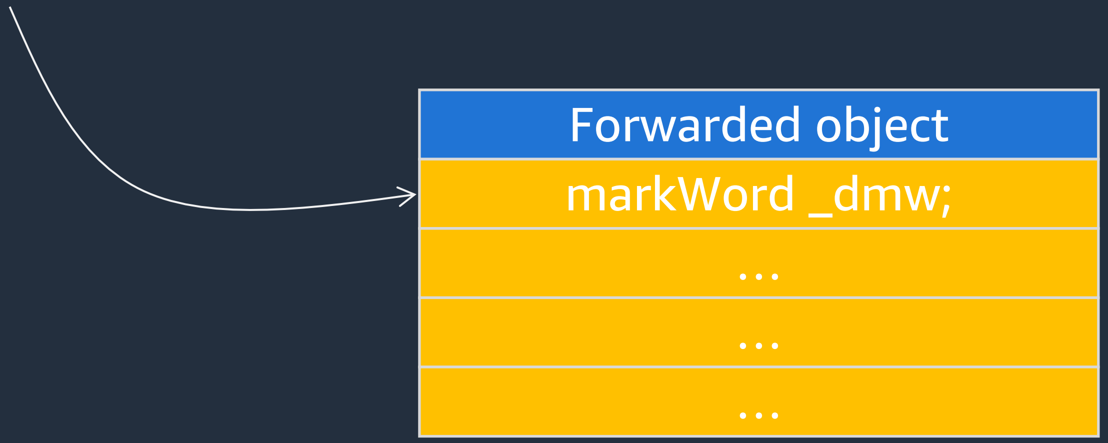
- Not a problem (yet)
- Oracle engineers are working on a solution

GC Forwarding

GC Forwarding

```
Mark Word (forwarded):

 64                                                           2 0
  [pppppppppppppppppppppppppppppppppppppppppppppppppppppppppppp11]
                 (Forwarding Pointer)                        (Tag)
```



GC Forwarding

GC Forwarding

	Serial	G1	Shenandoah	Parallel	ZGC
Normal	Copying Fwd	Copying Fwd	Copying Fwd	Copying Fwd	Fwd Table
Full GC	Sliding Fwd	Sliding Fwd	Sliding Fwd	Scissor GC	n/a

JEP 450

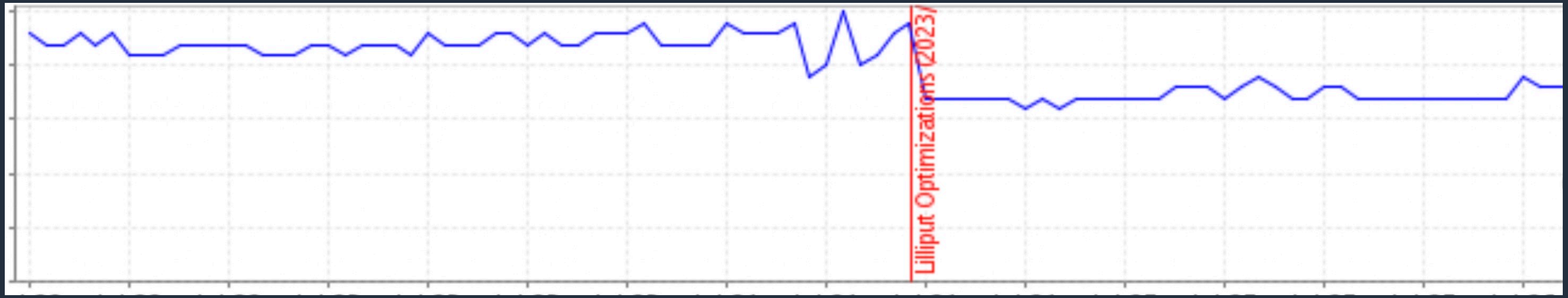
JEP 450: Compact Object Headers

- New lightweight locking in JDK21 (-XX:LockingMode=2)
- JEP 450: https://openjdk.org/jeps/450
- -XX:+UseCompactObjectHeaders

Wrapping up

-XX:+UseCompactObjectHeaders

https://openjdk.org/jeps/450



Tiny Classpointers